Escuela
Politécnica
Superior

Navegación de una apiladora industrial mediante aprendizaje por refuerzo

Grado en Ingeniería Robótica

Trabajo Fin de Grado

Autor:
Raúl Candela Arias
Tutor:
Tomás Martínez Marín

Universitat d'Alacant
Universidad de Alicante

Navegación de una apiladora industrial mediante aprendizaje por refuerzo

Autor

Raúl Candela Arias

Tutor

Tomás Martínez Marín

Departamento de Física, Ingeniería de Sistemas y Teoría de la Señal

Grado en Ingeniería Robótica

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Julio 2020

Preámbulo

Las razones principales que me han llevado a realizar este trabajo han sido mi desconocimiento del campo, así como el interés hacia los robots móviles, la navegación autónoma y las distintas ramas del aprendizaje automático.

Agradecimientos

A mis padres y mi pareja, sin ellos este trabajo no habría sido posible, me han ayudado incondicionalmente en los momentos más duros.

A mi compañera Ángela, con quien he compartido pupitre prácticamente desde el primer día hasta el último.

A mis compañeros y a la gran familia que hemos creado todos juntos, sin los que el grado habría resultado una experiencia muy distinta.

A mi tutor Tomás, por guiarme durante mi toma de contacto con el aprendizaje por refuerzo.

A mi familia, abuelos, tíos y primos por estar a mi lado y apoyarme desde siempre en todo lo que he hecho.

Y finalmente a mi hermano Carlos, que ha sido quien ha sufrido mis ganas de enseñar lo que aprendía.

A mi tío y a mi tía, que no han podido verme terminar.

Índice general

Índice de figuras

1. Introducción

Este proyecto consiste en la aplicación de técnicas de aprendizaje por refuerzo (*Reinforcement Learning*) en la navegación de una apiladora autónoma en un entorno real industrial. En particular, la navegación consistirá en el transporte de palés en un almacén compuesto por estanterías de cuatro niveles de altura. El algoritmo debe planificar la ruta óptima desde la carga de un palé en origen hasta la descarga del palé en destino. Para ello, debe seleccionar la secuencia óptima de acciones de avance y/o retroceso para las posibles orientaciones de llegada. Se realizarán tanto pruebas de simulación como pruebas experimentales en una apiladora real circulando en un entorno industrial.

Figura 1.1: Apiladora sobre la que se realizará el trabajo (Cortesía de QuixMind).

El aprendizaje por refuerzo permite al robot descubrir de forma autónoma un comportamiento óptimo a través de interacciones por prueba y error con su entorno. En lugar de detallar explícitamente la solución a un problema, en aprendizaje por refuerzo el diseñador de

una tarea de control aporta conocimiento en forma de una función que mide el rendimiento del robot tras realizar una acción (Kaelbling y cols., 1996).

Una solución convencional utilizada para implementar el planificador es mediante una búsqueda del camino de coste mínimo en un grafo. Para ello, se puede usar el algoritmo de Dijkstra para encontrar el mejor camino desde origen a destino a través de nodos intermedios. En la práctica, resulta muy costoso introducir todas las restricciones necesarias para resolver el problema, tal como orientaciones de origen y destino, cruces con limitaciones severas de espacio, preferencia de circulación en avance, etc. Como consecuencia, en el caso real de una planta industrial con 141 ubicaciones de palés (141 x 4 = 564 palés en total), el número de combinaciones es aproximadamente 20.000 caminos, con lo que se generan cientos de excepciones que hay que considerar de forma manual.

La técnica de aprendizaje por refuerzo proporciona un mecanismo más adecuado para introducir de forma declarativa los estados, las acciones y las restricciones del sistema en forma de transiciones posibles y función de recompensa. De este modo, se ha implementado el planificador sin requerir ninguna excepción y con la ventaja añadida de calcular todos los caminos óptimos para cada destino (permite la evitación óptima de obstáculos).

Se comparará la aplicación del aprendizaje por refuerzo con un enfoque más simple, una búsqueda del camino de coste mínimo de un grafo.Se seguirá la siguiente estructura:

- Marco Teórico.
  Donde se explicarán algunos conceptos necesarios para el entendimiento del documento y, en particular, la técnica de Q-learning.

- Objetivos.
  En este apartado se expondrá el objetivo principal del proyecto así como algunos hitos que se han establecido para lograrlo.

- Metodología.
  Donde se explicará cómo se resolverá el problema y los pasos que se han seguido hasta

su completa implementación, considerando pruebas de simulación y experimentación sobre una apiladora real.

- Desarrollo.
  En este apartado se explica el desarrollo y funcionamiento de la solución implementada, así como una pequeña comparación con otra posible solución basada en grafos.

- Resultados.
  En este apartado se presentan los resultados obtenidos de la aplicación de aprendizaje por refuerzo a la navegación de un apiladora en una planta industrial.

- Conclusión.
  Donde se hará un cierre del trabajo con un breve resumen de lo que se ha visto a lo largo de este y posibles líneas de trabajo.

2. Marco Teórico

A continuación se introducirán algunos conceptos necesarios para el correcto entendimiento del trabajo realizado.

Los grafos y el algoritmo de *Dijkstra* se mencionarán durante la explicación del motor de búsqueda basado en la búsqueda del camino mínimo. Seguidamente, se hablará sobre el aprendizaje automático y su clasificación como introducción al aprendizaje por refuerzo, que es el método utilidado para la implementación del planificador. Y por último, los procesos de decisión de Markov (*Markov Decision Process*) y el aprendizaje por refuerzo, incluyendo el *Q-Learning* y el principio de optimalidad de Bellman, se introducirán para la explicación del motor de navegación basado en aprendizaje por refuerzo.

2.1. Grafos

Un grafo $G = (V, E)$ consiste en un conjunto finito y no vacío de vértices V y un conjunto de aristas E. Si las aristas son pares ordenados (v, w) de vértices, entonces se dice que el grafo es dirigido.

Un camino en un grafo es una secuencia de aristas de forma $(v_1, v_2), (v_2, v_3), ..., (v_{n-1}, v_n)$. Decimos que el camino es de v_1 a v_n y de longitud n (Safavian y Landgrebe, 1991).

2.1.1. Algoritmo de Dijkstra

El algoritmo de *Dijkstra* resuelve el problema del camino de coste mínimo. Encuentra el camino de coste mínimo del nodo origen al nodo destino. El algoritmo busca el camino de coste mínimo entre los caminos del nodo origen a sus nodos adyacentes. El nodo terminal del

camino pasa a considerarse como un nodo intermedio. Entonces busca el siguiente camino de coste mínimo entre el nodo intermedio y sus nodos adyacentes. Las iteraciones continúan hasta que todos los nodos han sido transitados (Qing y cols., 2017).

2.2. Aprendizaje automático

El aprendizaje automático consiste en un sistema que modifica o adapta sus acciones de modo que estas se vuelven más precisas, donde la precisión se mide por como de bien encajan las acciones elegidas con las correctas (Marsland, 2015).
Consiste en programar ordenadores para optimizar el rendimiento según un cierto criterio usando datos o experiencia previa. Dado un modelo definido según una serie de parámetros, aprender consiste en la ejecución de un programa que cambia los parámetros del modelo usando los datos de entrenamiento o la experiencia previa (Alpaydin, 2020).

2.2.1. Paradigmas del aprendizaje automático

En el aprendizaje automático es posible identificar varios paradigmas según los criterios que se siguen, como por ejemplo, según el tipo de selección y adaptación que un sistema realiza sobre la información, o según el tipo de estrategia y las ayudas que recibe el sistema de aprendizaje (Moreno y cols., 1994). Para este proyecto solo será necesario el último enfoque mencionado.

2.2.1.1. Según el tipo de estrategia y las ayudas que recibe el sistema

- Supervisados

  Para la aplicación de técnicas de aprendizaje supervisado se parte de un conjunto de datos iniciales (entradas) para los que se observa una serie de resultados (salidas) para una serie de eventos determinados. A partir de estos datos, es posible construir un modelo de predicción, que permitirá predecir los resultados que se obtendrán para un nuevo evento (Cardenas y cols., 2015).

- No supervisados

  El aprendizaje no supervisado consiste en hallar características, regularidades, correla-

ciones o categorías en los datos de entrada. En algunos casos, la salida representa el grado de similitud entre la información que se le está presentando en la entrada y la que se le ha mostrado en el pasado. En otro caso podría realizar un establecimiento de categorías, indicando con la salida a qué categoría pertenece la información presentada como entrada (Peláez, 2012).

- Mediante refuerzos

  El aprendizaje por refuerzo se basa en aprender qué hacer, por ejemplo, cómo mapear las situaciones a acciones, de forma que se maximice la recompensa numérica. Al agente no se le dice que acciones debe tomar, sino que debe descubrir la que le otorga una recompensa mayor a través de prueba y error. Las acciones pueden afectar a las recompensas futuras, además de la actual. Estas dos características (búsqueda por prueba y error y recompensas tardías) son las más importantes del aprendizaje por refuerzo (Sutton y Barto, 2018). Este término será ampliado a continuación.

2.3. Procesos de decisión de Markov

Un proceso de decisión de Markov es un sistema estocástico caracterizado por una 5-tupla $\langle \mathbf{S}, \mathbf{A}, A, p, g \rangle$, donde: $\mathbf{S}$ es un conjunto finito de estados discretos y $\mathbf{A}$ son las acciones de control. Mapeando $\mathbf{A}$: $\mathbf{S} \rightarrow p(\mathbf{A})$ es la función de disponibilidad que representa un conjunto de acciones disponibles para cada estado. La función de transición viene dada por p: $\mathbf{S} \times \mathbf{A} \rightarrow \Delta(\mathbf{S})$, donde $\Delta(\mathbf{S})$ es el conjunto de todas las distribuciones de probabilidad sobre $\mathbf{S}$. $p(y \mid x, a)$ denota la probabilidad de llegar al estado y después de ejecutar la acción $a \in A(x)$ en el estado x. La función de coste inmediato está definida por g: $\mathbf{S} \times \mathbf{A} \rightarrow \mathbf{R}$, donde $g(x, a)$ es el coste de tomar la acción a en el estado x.

Una política de control determina la acción que tomar en cada estado. Una política determinista π: $\mathbf{S} \rightarrow \mathbf{A}$, es simplemente una función de estados a acciones de control. Una política estocástica π: $\mathbf{S} \rightarrow \Delta(\mathbf{A})$ es una función de estados a distribuciones de probabilidad de acciones de control. Denotamos la probabilidad de ejecutar la acción a en el estado x con $\pi(x)(a)$ o $\pi(x, a)$ para acortar (Valeev y Kondratyeva, 2016).

2.4. Aprendizaje por refuerzo

El aprendizaje por refuerzo es el problema al que se enfrenta un agente que debe aprender a través de prueba y error con interacciones en un entorno dinámico. Tiene una fuerte similitud a su epónimo en psicología, pero difiere considerablemente en el uso de la palabra *refuerzo*. Se puede pensar en ello como una clase de problemas en vez de como en un conjunto de técnicas.

Hay dos estrategias principales para resolver problemas de aprendizaje por refuerzo. La primera es buscar en el espacio de comportamientos para encontrar uno que funcione bien en el entorno. La segunda es usar técnicas estadísticas y métodos de programación dinámica para estimar la utilidad de tomar acciones en los estados (Kaelbling y cols., 1996).

2.4.1. Q-Learning

Q-Learning es uno de los métodos de aprendizaje por refuerzo más populares, ya que puede resolver problemas de optimización sin modelo y tiene una formulación simple. El conocimiento se guarda en una tabla de consulta que contiene la estimación de la recompensa por llegar al destino en cada situación o estado. La recompensa acumulada de cada par estado-acción $Q(s, a)$ se actualiza con la ecuación

$$\Delta Q(s, a) = \alpha(r + \lambda \, max_{a'} Q(s', a') - Q(s, a)),$$

donde Q es el valor esperado por realizar la acción a en el estado s, r es la recompensa, α es el ratio de aprendizaje que controla la convergencia y λ es el factor de descuento. Este factor de descuento da más valor a las recompensas inmediatas que a las futuras. La acción a con el mayor valor de Q en el estado s es la mejor política hasta el momento t (Martinez-Marin y Duckett, 2005). Según el tipo de recompensa (función de coste) la solución será de tiempo mínimo, energía mínima, etc. En nuestro caso hemos optado por una solución óptima en tiempo mínimo.

2.4.2. Principio de optimalidad de Bellman

La ecuación de entrenamiento del algoritmo de Q-learning expresada anteriormente se basa en el principio de optimalidad de Bellman, el cual se puede enunciar del modo siguiente: "Una política óptima tiene la propiedad de que independientemente del estado y acción inicial, las decisiones restantes deben constituir una política óptima con respecto al estado resultante de la primera acción (Bellman, 1957)".

Por tanto, la estrategia para propagar la política óptima se puede expresar de forma recurrente partiendo del estado final (destino), cuya recompensa es conocida. La aplicación de la ecuación entre estados vecinos de forma recurrente propaga la ruta óptima desde el destino hasta todos los orígenes posibles.

3. Objetivos

El objetivo principal de este proyecto es la aplicación de técnicas de aprendizaje por refuerzo para que una apiladora sea capaz de navegar de forma autónoma por una nave industrial real.

También se establecieron los siguientes hitos para seguir un desarrollo más modular del trabajo:

- Crear el espacio de estados de la nave industrial.
  Dado que una gran cantidad de la planta industrial compuesta por estanterías y pasillos no es transitable, se decidió crear estados discretos asociados a los palés en vez de crear una partición de estados densa en función de la posición x, y, θ posible del robot. De este modo, la navegación se encamina a la circulación por pasillos y los cruces que aparecen entre estanterías, reduciendo de forma sustancial la cantidad de estados necesarios.

- Definición de las restricciones del planificador para lograr el resultado deseado.
  Tal como se ha comentado, una parte fundamental para el correcto funcionamiento del aprendizaje por refuerzo es la definición de las transiciones y la función recompensa para, por ejemplo, evitar ir en retroceso en la medida de lo posible, o evitar dar vueltas innecesarias para llegar al objetivo en avance. Por ello, es necesario buscar un equilibrio entre costes o recompensas y prohibición de ciertas transiciones.

- Generar los objetivos a partir de la ruta óptima de salida del planificador basado en aprendizaje por refuerzo.
  Una vez generado el camino óptimo, es necesario pasar de las acciones en el espacio de estados a una consigna de control de la máquina con la intención de alcanzar objetivos intermedios a lo largo del recorrido planificado. Por ello, se segmenta el camino en los

tramos de nodo a nodo, estableciendo los objetivos necesarios para seguir el trayecto y, además, teniendo en cuenta los cambios de orientación planificados.

- Implantación en un sistema real, pruebas experimentales y calibración.
  Debido a las diferencias entre la máquina simulada y la máquina real, el correcto funcionamiento del sistema en simulación no asegura que funcione igual en real. Por esto se ha realizado una serie de pruebas en la nave para calibrar las distancias de giro y asegurar el correcto funcionamiento del sistema.

4. Metodología

4.1. Estudio del problema

El problema consiste en la planificación óptima del transporte de palés en una nave industrial, donde se contemplan varias posibilidades para su solución:

- Navegación mediante bandas magnéticas o líneas pintadas en el suelo.
- Navegación mediante un motor de búsqueda basado en la búsqueda del camino mínimo.
- Navegación mediante aprendizaje por refuerzo.

Para la primera solución es necesaria una modificación demasiado costosa en la nave. En cambio, para las dos últimas es necesario un sistema de percepción basado en *SLAM* (Simultaneous Localization and Mapping), capaz de localizarse sin marcas independientemente de donde se encuentre en la nave.

4.2. Método a utilizar

La mejor opción es la navegación basada en aprendizaje por refuerzo debido a que el problema se puede plantear como un Proceso de Decisión de Markov, donde las transiciones entre estados pueden ser generadas de forma automática. En particular, dado que el número de estados ha sido reducido, hemos empleado la técnica de *Q-Learning*. Esto conlleva una serie de beneficios como la posibilidad de planificar la orientación inicial y final, la de planificar un cambio de orientación a mitad del recorrido o la certeza de que el camino planificado será óptimo. El funcionamiento es el siguiente:

1. Un operario solicita una tarea.

2. La tarea se segmenta en tramos (ubicación actual-origen y origen-destino).

3. El planificador recibe un segmento de la tarea y genera los objetivos intermedios a seguir.

4. Los objetivos se le pasan al controlador y este hace que la apiladora vaya transitando por ellos en el orden especificado.

5. El robot realiza la carga o la descarga.

6. En caso de que queden más tareas pendientes, estas se encadenan sin pasar por *HOME* para minimizar el tiempo de transporte. En caso contrario, la máquina vuelve a su posición *HOME*, cerca de la estación de carga (ver figura 5.1).

4.3. Pruebas de simulación

Para la resolución de este problema, en primer lugar se debe trabajar con un entorno simulado para poder probar miles de tareas en tiempos de simulación asumibles. Las pruebas de planificación se realizan con un test donde se simula la navegación desde cada posible origen a cada posible destino en orden aleatorio. Más tarde, se contabiliza el número de errores que se producen y se divide entre el número total de tareas realizadas, con lo que se obtiene el ratio de $\frac{TareasErradas}{TareasRealizadas}$.

El test consiste en comprobar si durante la realización de las tareas la máquina simulada entra en zonas definidas por un perímetro como prohibidas, es decir, sale de la zona especificada para navegación segura. Esta zona se puede apreciar en la figura 4.1 delimitada por el perímetro verde.

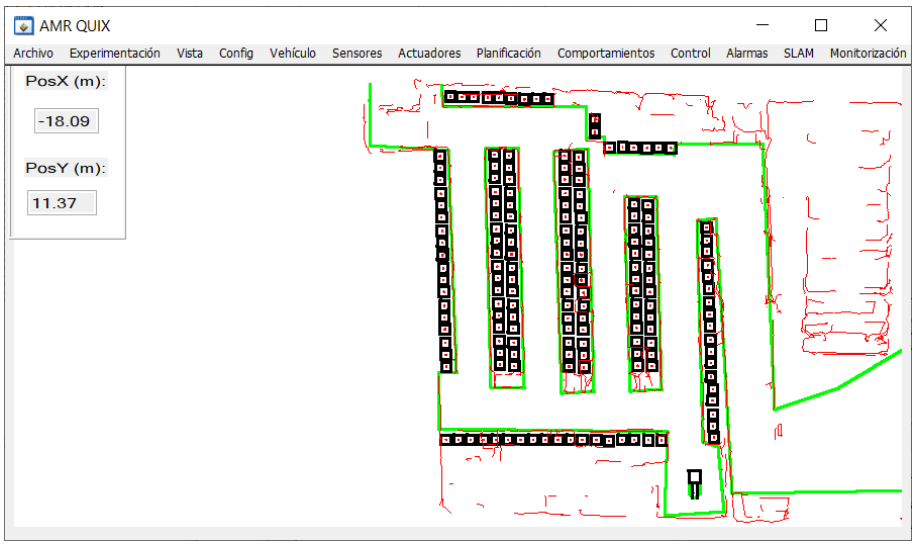
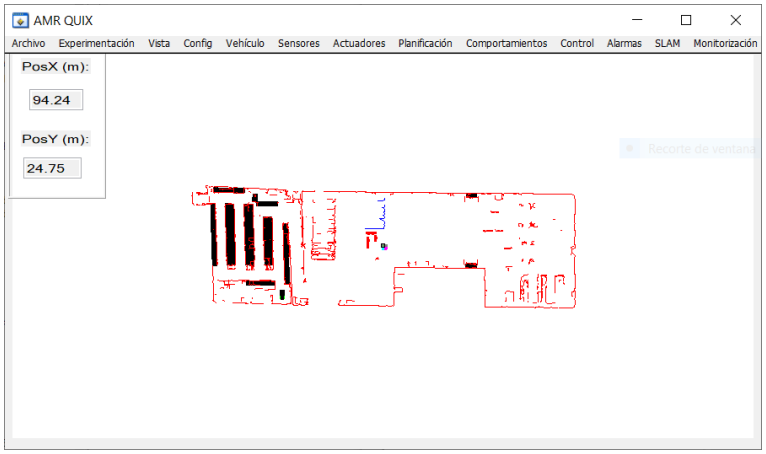
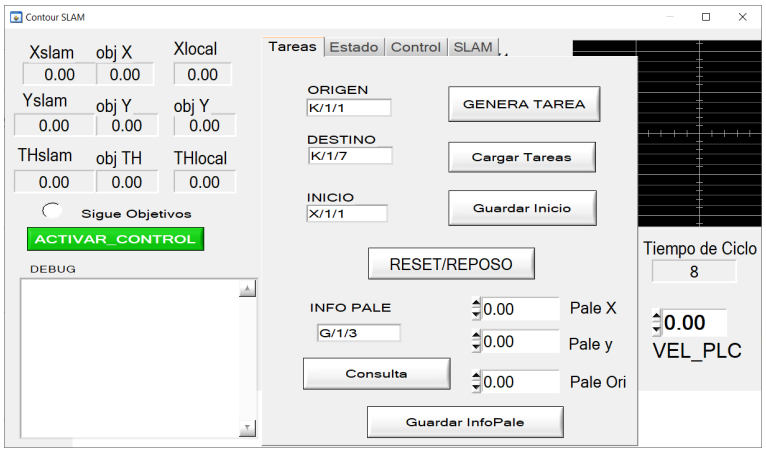


Figura 4.1: Zona prohibida.

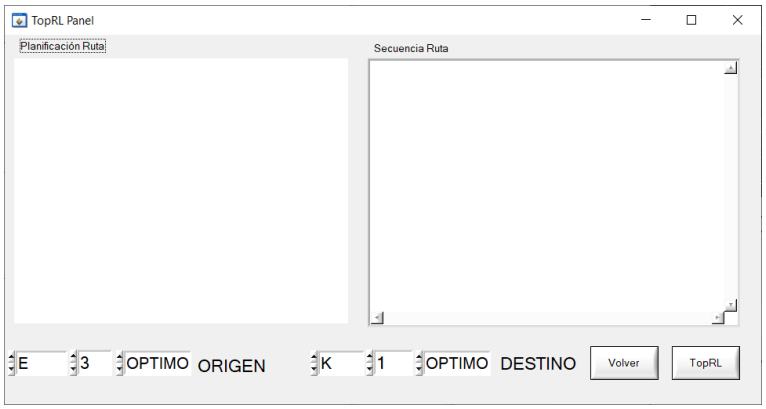
El entorno consiste en tres ventanas, una con el mapa donde se dibuja la apiladora en su posición simulada, una de control donde se puede lanzar una tarea para ser simulada o cargar un fichero de tareas, y una con el planificador para ver la salida de este cuando se lanza una tarea o probarlo sin llegar a simular la tarea completa, probando así solo el planificador.



(a) Mapa.



(b) Panel de control.



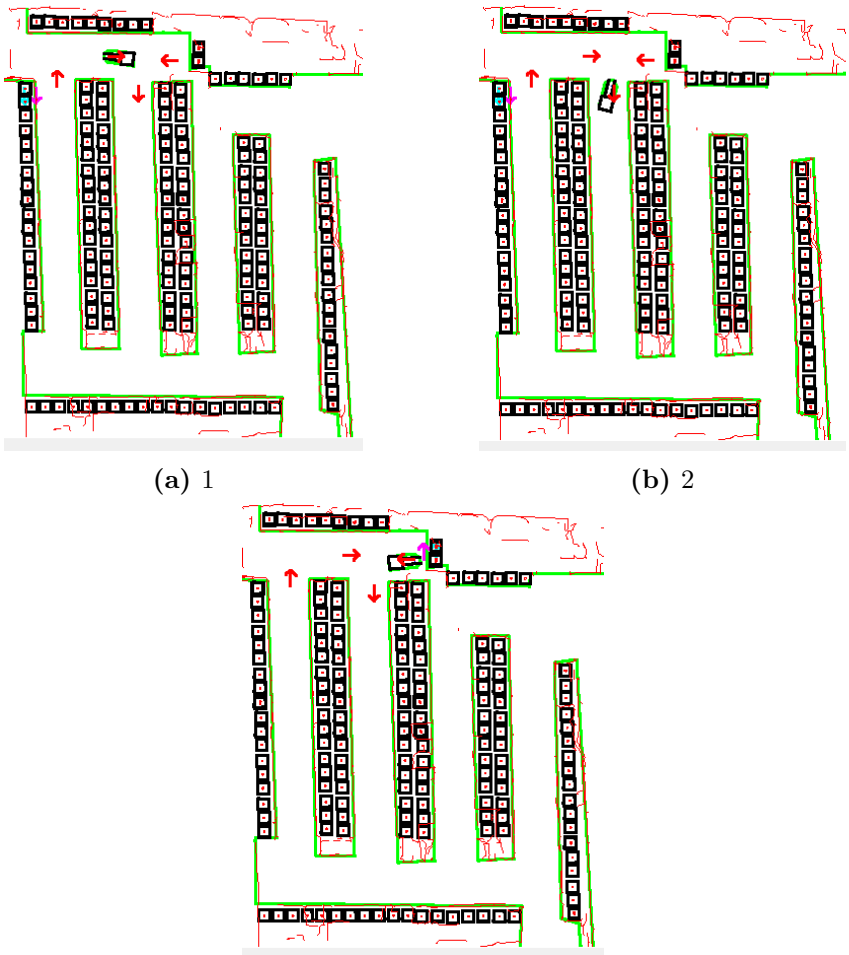
(c) Salida del planificador.

Figura 4.2: Entorno de simulación.

4.4. Pruebas en el robot real

Una vez que el planificador ha empezado a funcionar de forma robusta en simulación (alrededor de 0.01 $\frac{TareasErradas}{TareasRealizadas}$), se inician las pruebas con el robot real, ya que el movimiento simulado es distinto del movimiento del robot real. Por ejemplo, el robot simulado necesita menos espacio para girar 90º y los retardos producidos por el PLC no se simulan. Por esto, ha sido necesario ajustar algunos valores para los giros, obteniendo un resultado muy parecido al de simulación. Se ha logrado realizar con éxito los cambios de orientación como el mostrado en la figura 4.3.

De este modo, se ha realizado con éxito el diseño, implementación y puesta en marcha de la planificación de ruta óptima basado en aprendizaje por refuerzo.



(a) 1 **(b)** 2

(c) 3

Figura 4.3: Resolución de un cambio de orientación.

5. Desarrollo

5.1. Planificación

El entorno en el que se desarrolla esta aplicación consta de palés, estanterías, pasillos y nodos. Un palé es una posición establecida en x, y y θ, una estantería es una agrupación de palés, un pasillo es el hueco entre dos estanterías y un nodo es un cruce entre dos o más pasillos.

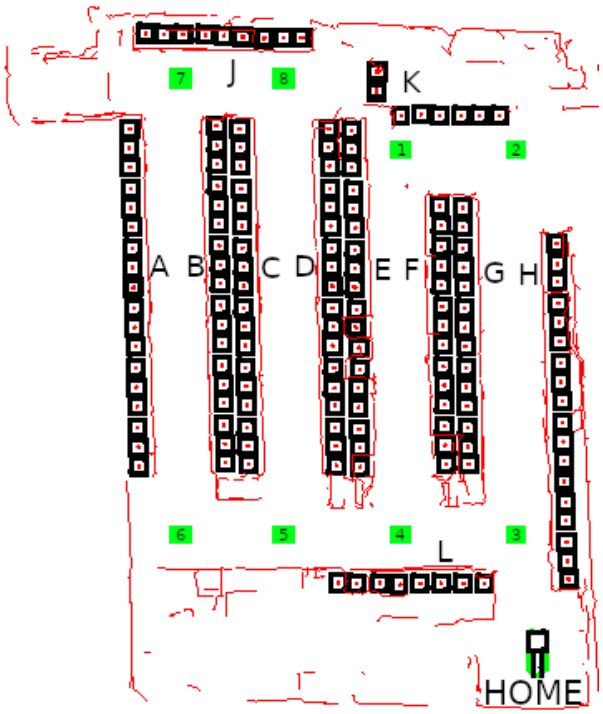


Figura 5.1: Mapa de la nave.

Como podemos ver en la figura 5.1, los palés son las cajas negras, las estanterías son las

agrupaciones de palés encerradas por un marco rojo, los pasillos son el hueco libre que queda entre ellas por el que la apiladora se moverá y los nodos son las cajas verdes donde se cruzan dos o más pasillos. Tanto las estanterías como los nodos tienen un identificador único, las estanterías de *A* a *K* sin pasar por *I* y los nodos de *1* a *8*.

Si se prueba con un motor de búsqueda basado en encontrar el camino de coste mínimo de un grafo (*Dijkstra*), cuyos vértices son los nodos y las aristas los pasillos, este solo considera que la máquina se mueve a lo largo de los pasillos, pero no los cambios de avance a retroceso y viceversa. Esto se traduce en que recibe una entrada (palé origen/palé destino) y produce una salida (vector de nodos). Por ejemplo, para la entrada $J/1/9 \rightarrow K/1/6$ produce la salida $[8, 5, 4, 1]$

Cuando se especifica un destino, a causa de que no contempla el cambio de avance a retroceso, la única manera de asegurar la llegada en retroceso es hacer todo el movimiento en retroceso. Esto no es factible debido a que ir en retroceso representa un grave riesgo para una persona en caso de fallo de las seguridades, por lo que hay que intentar minimizar, en la medida de lo posible, las distancias que se recorren con las horquillas por delante.
También se podría optar por no ir nunca en retroceso, pero hay palés cercanos a esquinas a los que solo se puede acceder en retroceso, por ejemplo, en la figura 5.2 se ve como al intentar llegar al destino (palé en amarillo) en avance choca con la estantería de al lado.

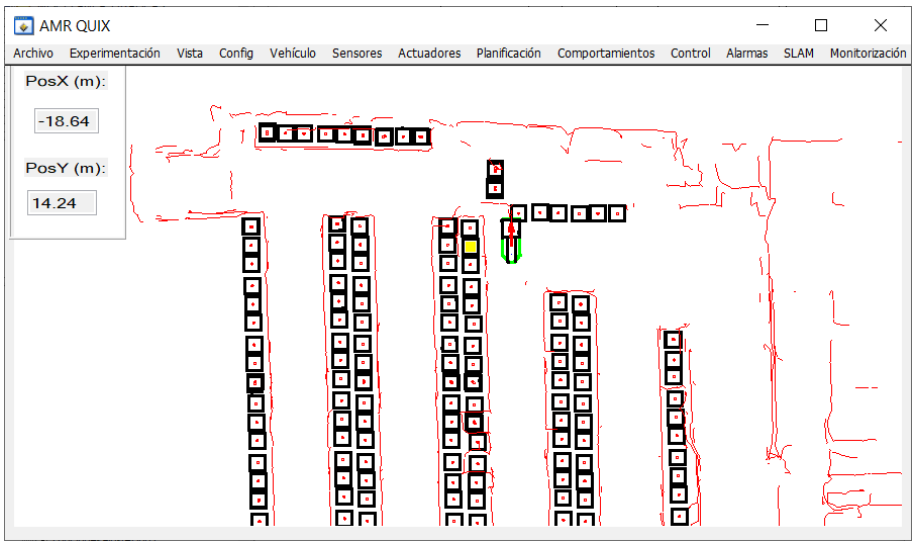


Figura 5.2: Choque con la estantería por llegada en orientación incorrecta.

Debido a la simplicidad de este motor de búsqueda, la planificación que se puede realizar a veces es insuficiente y muchos errores deben ser solucionados durante la generación de objetivos. A veces, llegando incluso a añadir varios objetivos fijos necesarios para un único par origen/destino o para una familia de trayectorias. Esto complica solucionar los problemas de la generación de objetivos debido a la mezcla no uniforme de código genérico para su creación, con código específico para una familia de tareas y para una única tarea.

Por estas limitaciones mencionadas anteriormente, se contempla la opción de crear un motor de búsqueda basado en un **MDP** (*Markov Decision Process*). Donde un algoritmo de aprendizaje por refuerzo, descubre la forma óptima de llegar al estado solución combinando una serie de posibles acciones tanto en avance como en retroceso, y minimizando el coste numérico acumulado.

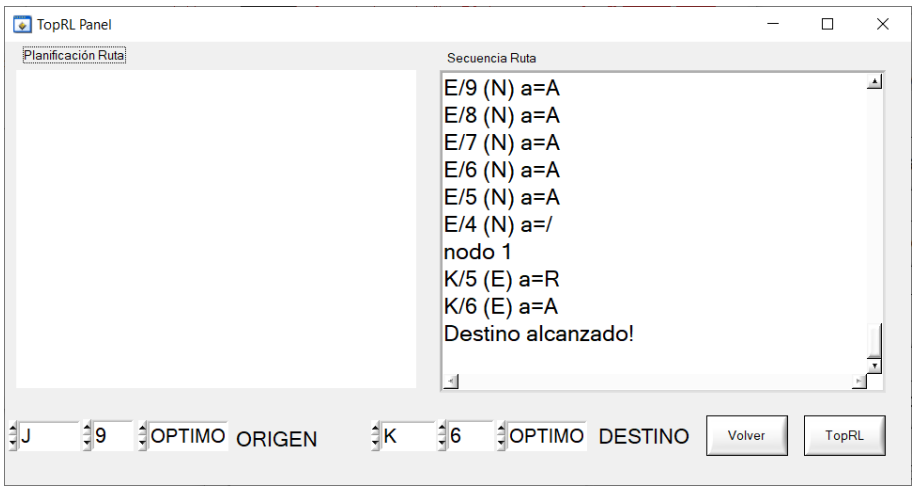


Figura 5.3: Salida del planificador

Este motor de búsqueda es mucho más flexible y potente. Para la misma entrada (palé origen/palé destino) produce una sucesión de estados y acciones (ver figura 5.3 y anexo A) de la que se puede extraer:

- En qué orden pasar por los nodos.
- Qué acción nos lleva al siguiente estado.
- Qué orientación tiene la apiladora en cada estado.

También permite calcular la planificación en función de la orientación inicial. Cuando se coge o se deja un palé, se puede salir en cualquiera de las tres orientaciones, pero en caso de sufrir una parada al encontrar un objeto a mitad del camino, se puede recalcular la ruta teniendo en cuenta que debe empezar con la orientación con la que paró.

El espacio de estados en vez de estar dividido en una rejilla según x, y y θ, está dividido a razón de estado por ubicación de palé con un total de 141 estados. Para cada estado, también se tiene en cuenta la orientación, que debido a que no hay estanterías oblicuas, está discretizada a *Norte*, *Oeste*, *Sur* y *Este*, por lo que para cada estado asociado a un palé tenemos 4 subestados.

Para la creación del espacio de estados total, se encadena de forma manual en un vector los códigos de generación del espacio de estados de todos los pasillos. El espacio de estados de un pasillo se genera de forma semiautomática a partir de los siguientes elementos en el orden que se expresa a continuación:

1. Un nodo.
2. El identificador del palé en el que empieza el pasillo.
3. Un código que indica que se interpolarán los estados intermedios.
4. El identificador del palé en el que acaba el pasillo.
5. El nodo al que se llega siguiendo el pasillo.

Esto genera todos los estados asociados a cada palé del pasillo y los conecta entre sí en función de la acción realizada. Tenemos seis acciones distintas, tres en avance y tres en retroceso. Estas son:

- Avance izquierda.
- Avance.
- Avance derecha.
- Retroceso izquierda.
- Retroceso.
- Retroceso derecha.

Las transiciones internas del pasillo generadas automáticamente como consecuencia de la declaración en el vector mencionado anteriormente se enlazan entre sí con las acciones de *Avance* y de *Retroceso*. En el caso de las transiciones pertenecientes a las intersecciones, deben ser declaradas manualmente, así como la eliminación de las transiciones no deseadas debido a las limitaciones del entorno. Esto se hace por medio de una configuración en la que se declaran las intersecciones según su tipo:

- Cruce *L*.

- Cruce T.
- Cruce de cuatro caminos.

Los cruces de tipo L son solo entre dos pasillos y es el que se da en los nodos uno y seis, por ejemplo. Los cruces de tipo T involucran a tres pasillos, y es el que se da en los nodos cuatro y cinco, por ejemplo (ver figura 5.1). Finalmente, los cruces de tipo *cuatro caminos*, son una intersección que involucra 4 pasillos, en este proyecto no hay ninguno de este tipo.

En cuanto a la búsqueda del camino óptimo se pone una recompensa muy elevada en los estados asociados al palé destino. En caso de ser necesario asegurar la orientación de la llegada, solo se pondrá la recompensa en el estado asociado a la orientación deseada.

También tenemos recompensas locales asociadas a la acción realizada, estas están pensadas para penalizar los giros y el movimiento en retroceso, y asegurar el tiempo mínimo. Esta solución es válida debido a que las acciones representan más o menos la misma cantidad de desplazamiento sobre x, y y θ. Los valores de las acciones son los siguientes:

- Avance izquierda: -2.
- Avance: -1.
- Avance derecha: -2.
- Retroceso izquierda: -5.
- Retroceso: -4.
- Retroceso derecha: -5.

Se podría haber optado por una función de coste donde este viene dado por la estimación del tiempo que se tardará en realizar la acción. Por ejemplo, la distancia Manhattan dividida por la velocidad media del movimiento, debido a que los movimientos en retroceso son más lentos por seguridad, esta función de coste también aseguraría llegar al destino en tiempo mínimo y con movimientos de avance siempre que sea posible.

A la hora de propagar los valores de Q, para cada estado se toman todas las acciones posibles, y en función de la recompensa obtenida, se actualiza el valor de Q del estado siguiendo la regla de actualización de Q.

$$Q(s_t, a_t) = (1 - \alpha) * Q(s_t, a_t) + \alpha * (r_t + \gamma * max\ Q(s_{t+1}, a_{t+1}))$$

Donde α es el índice de aprendizaje, r_t la recompensa obtenida por transitar del estado s_t a s_{t+1} y γ es el factor de descuento que cambia la importancia de las recompensas futuras por la de las recompensas inmediatas. En este caso el índice de aprendizaje y el factor de descuento serán uno, ya que como siempre se recibe una recompensa ligada a la acción, el valor de Q no variará después de la primera actualización.

Por cada iteración de todos los estados, se actualizan los valores de Q a distancia de un estado de los previos valores actualizados. Por ejemplo, para $t = 1$ se actualizan los estados vecinos al estado solución, para $t = 2$ se actualizan los estados vecinos a los que se actualizaron con $t = 1$, y así hasta haber propagado a todos los estados el valor de Q.

Llegados a este punto, para obtener el camino óptimo, basta con ir de origen a destino por los estados con un mayor valor de Q. Esto asegura que se cumple el principio de optimalidad de Bellman, ya que cuando se propagaba el valor de Q, se generaban caminos óptimos de los estados actualizados al estado destino utilizando los caminos óptimos previos.

5.2. Generación de objetivos

Para el movimiento de la apiladora, se colocan unos elementos llamados *Objetivos* que definen:

- Posición en x, y y θ por la que va a pasar la apiladora.
- Velocidad que llevará la apiladora desde el anterior objetivo hasta este.
- Acción que realizará la apiladora al llegar al objetivo, puede ser *Carga*, *Descarga* o

Nada.

- En caso de *Carga* o *Descarga*, el identificador del palé a coger o del hueco donde se va a dejar.
- Dirección del movimiento, puede ser *Avance* o *Retroceso*.

Para hacer una trayectoria, el algoritmo de generación de objetivos recibe como entrada la salida del planificador y produce como salida un vector de *Objetivos* a los que se irá según el orden del vector. Por esta razón, el algoritmo de generación de objetivos asociado al planificador basado en grafos es mucho más simple debido a que recibe mucha menos información como entrada. Mientras que el algoritmo asociado al planificador basado en aprendizaje por refuerzo es más complejo y permite usar reglas más potentes y obtener un mejor resultado.

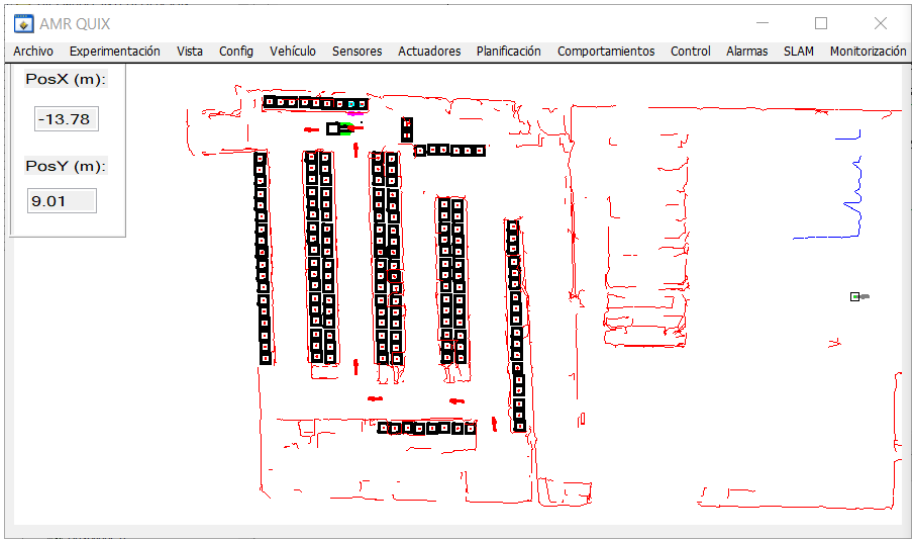


Figura 5.4: Visualización de los objetivos generados para una tarea.

Debido a que el modelo de movimiento de la apiladora es el de un vehículo Ackerman, tiene dificultades para desplazarse de forma perpendicular a su eje de movimiento, es por esto que si se ponen dos objetivos muy cercanos desplazados en esta dirección tendría graves problemas para lograr llegar a la posición del segundo. Por esta razón se separa el camino en

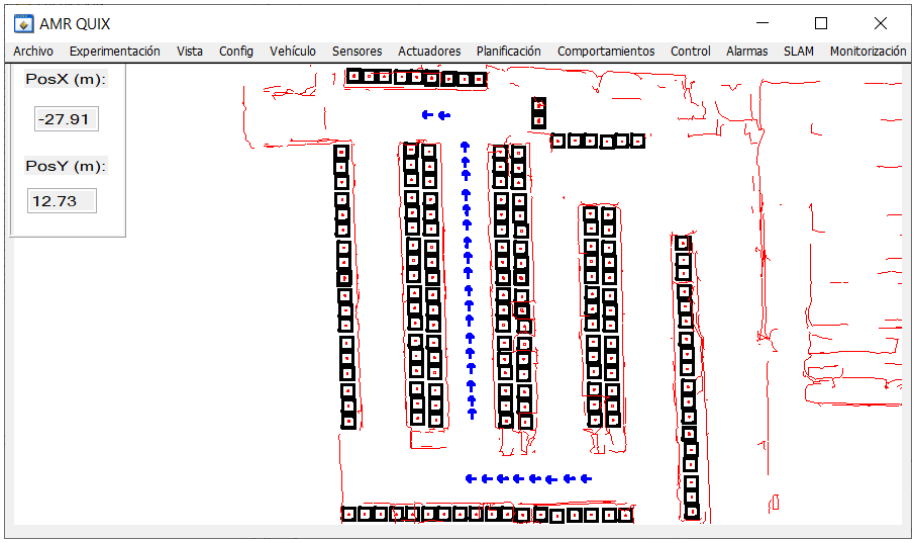
tramos, y se ponen dos objetivos por pasillo a atravesar separados de los nodos del siguiente tramo una distancia calibrada para que no choque con la estantería durante el giro.

6. Resultados

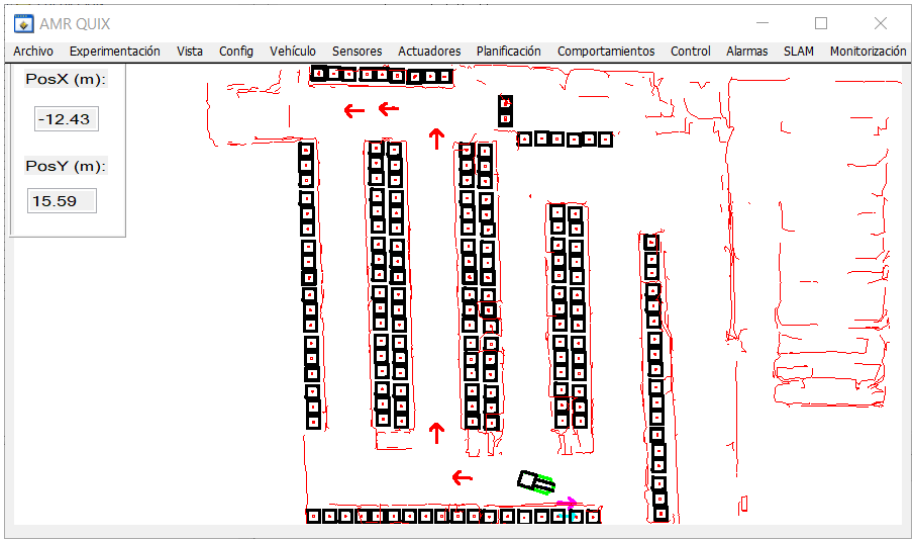
En este apartado se van a explicar algunos casos prácticos en los que la apiladora va de una posición a otra.

6.1. Resultados de simulación

En la simulación se puede probar tanto la parte de planificación como la parte de generación de objetivos, para ambas partes se dibujan unas flechas que indican por dónde pasará la apiladora, pero tienen finalidades distintas. Las flechas del planificador sirven para dibujar un camino estimado del recorrido del espacio de estados, mientras que las flechas que se pintan en la generación de objetivos son los propios objetivos, puntos definidos por los que pasará la apiladora.



(a) Flechas de planificación.



(b) Flechas de objetivos.

Figura 6.1: Diferencias entre flechas de planificación y de objetivos para una misma tarea.

Para la parte de la planificación, cuando se propaga el valor de Q, obtenemos todos los posibles orígenes para el destino seleccionado (cuadro amarillo), en la figura 6.2 podemos ver como según aumenta el valor de Q, las flechas se han pintado de un color más cálido.

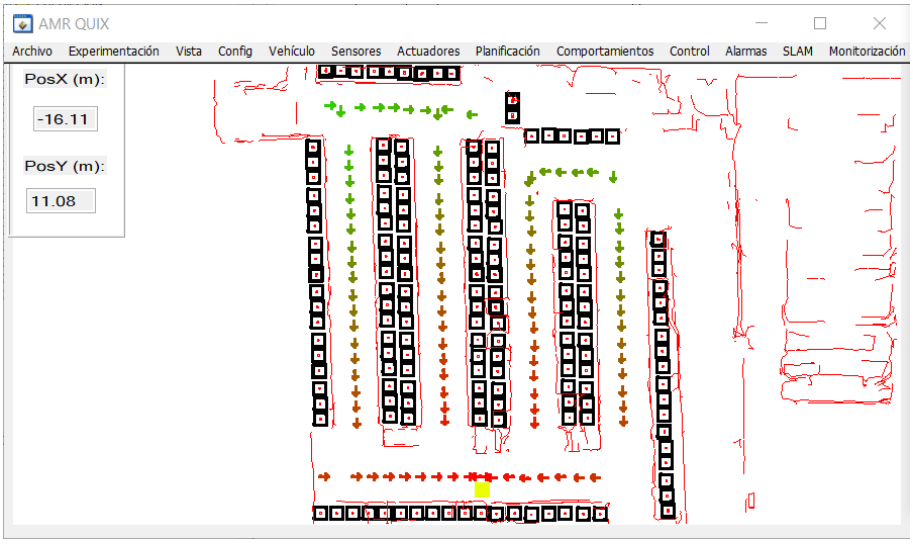
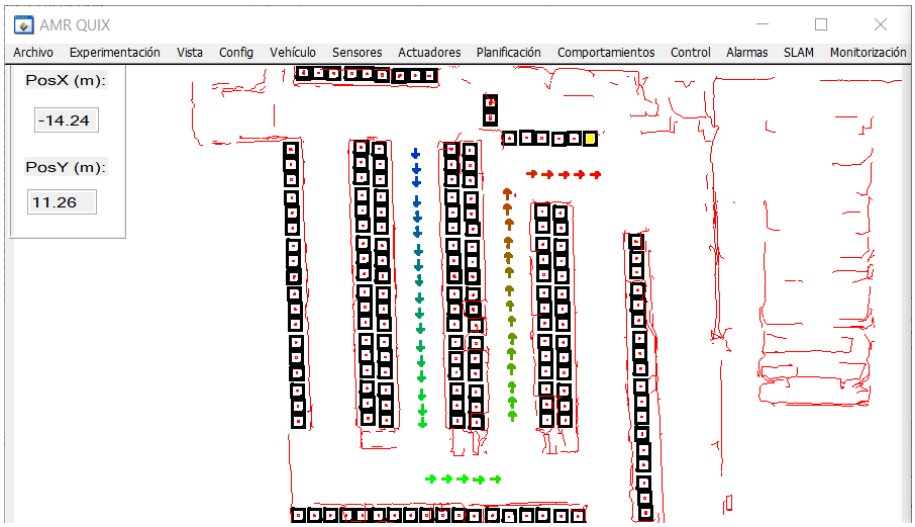
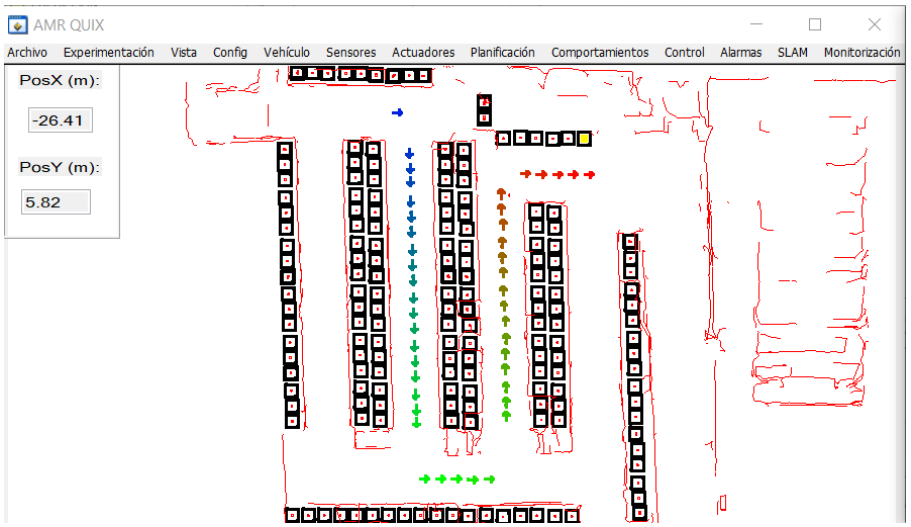


Figura 6.2: Posibles orígenes para el destino L/1/8.

En la figura 6.3 podemos ver que los caminos generados cumplen el principio de optimalidad de Bellman, el camino óptimo está formado por una acción que nos lleva a un nuevo estado y el camino óptimo desde este nuevo estado. Por ejemplo, el camino de $C/1/1$ a $K/1/1$ forma parte del camino de $J/1/7$ a $K/1/1$.



(a) Tarea C/1/1 → K/1/1.



(b) Tarea J/1/7 → K/1/1.

Figura 6.3: Composición de caminos.

Gracias a las flechas de la planificación, también se pueden ver de forma muy clara los cortes de los flujos del sentido, por ejemplo, en la figura 6.4 podemos ver cómo a la altura del punto verde cambia el sentido del camino óptimo de pasar por enfrente de la estantería L a pasar por enfrente de la estantería K (ver figura 5.1).

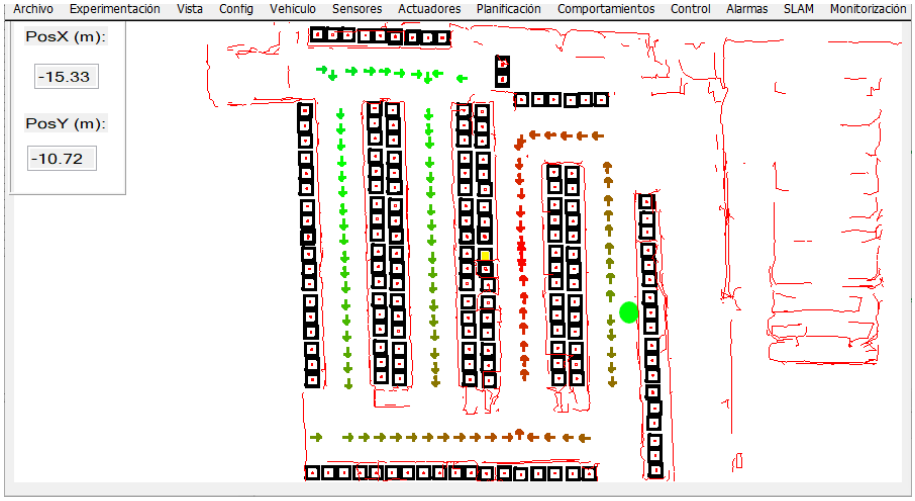
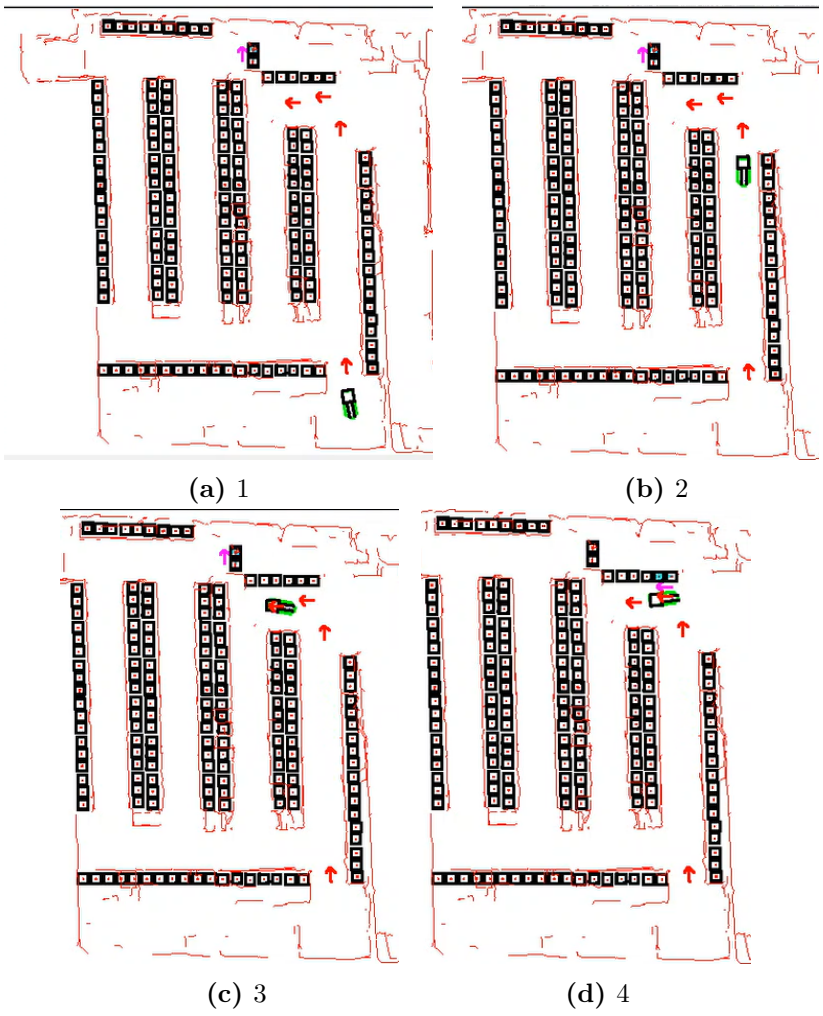


Figura 6.4: Posibles orígenes para el destino E/1/10 (en amarillo).

Para la generación de objetivos, se pasa la salida del planificador a un algoritmo que produce los objetivos mínimos y necesarios para asegurar la llegada de la apiladora a su destino en régimen permanente, ya que en caso de llegar con error en x, y o θ puede chocar al encararse con la estantería de detrás.

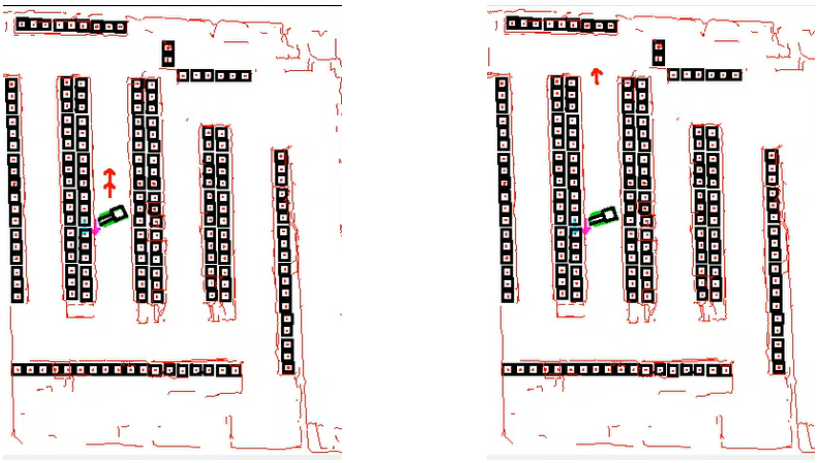
Para asegurar la llegada en régimen permanente, el mecanismo principal es pasar de largo el destino y volver a él en retroceso. Esto se puede ver en la figura 6.5, y se suele realizar cuando el destino está muy cercano a un giro, o al origen (ver figura 6.6).



(a) 1 **(b)** 2

(c) 3 **(d)** 4

Figura 6.5: Objetivos asegurando régimen permanente.



(a) Destino lejano al origen. **(b)** Destino cercano al origen.

Figura 6.6: Comparación destino cercano y lejano a origen.

6.2. Resultados experimentales

En este proyecto de fin de carrera, se ha tenido acceso a un robot apiladora instalado en una planta industrial. Por ese motivo, se han podido hacer pruebas experimentales del planificador basado en aprendizaje por refuerzo en una fábrica con todas las restricciones habituales en entornos industriales y se ha podido implementar el sistema en el robot real. Para ello se han realizado una serie de pruebas durante las que se produjeron algunos errores no simulables como:

- Errores de visión a la hora de la carga cuando se detectaba que había espacio insuficiente.
- Errores de seguridades debido a partículas de polvo en suspensión que hacían saltar los láseres de seguridad.
- Errores de conexión a la hora de comunicarse a través de la red con la aplicación del cliente.
- Errores de generación de objetivos, ya que al intentar minimizar los movimientos innecesarios, cuando se planificaba un cambio de orientación, la máquina real no era capaz de llegar al régimen permanente en una distancia tan pequeña como la máquina simulada.

Sabiendo esto último de antemano, se optó por una implementación donde todas las distancias necesarias para alejarse y lograr el régimen permanente en las diferentes casuísticas estaban parametrizadas, y bastó con un día de pruebas para dejar calibrada la apiladora funcionando en la nave del cliente siendo capaz de realizar entorno a 19600 trayectorias distintas con código completamente genérico, sin ninguna parte dedicada a una familia de trayectorias y sin objetivos fijos, como pasaba con el motor de navegación basado en grafos. Esto implica que para una nueva nave bastaría con colocar los nodos, crear el espacio de estados y en caso de haber cambiado de máquina, reajustar las distancias calibradas para lograr el régimen permanente para que se ajusten al nuevo modelo, siendo así capaz de navegar.

Estas trayectorias se han probado de forma exhaustiva asegurando que no había ningún error durante la simulación antes de dejar la apiladora operativa en la nave del cliente, y no habiendo recibido ningún reporte por parte de este.

Figura 6.7: Apiladora en navegación (Cortesía de QuixMind).

Figura 6.8: Apiladora cargando un palé (Cortesía de QuixMind).

7. Conclusiones

Como hemos podido apreciar a lo largo del trabajo, existían dos posibles métodos para la navegación autónoma de la apiladora. En el primer caso, nos encontramos con un motor de búsqueda basado en encontrar el camino mínimo de un grafo, en el que no se podía asegurar la orientación de llegada e impedía planificar un cambio de orientación ya que solo tiene en cuenta que la apiladora pasa por un pasillo y no la orientación de paso, esto obliga a hacer todo el movimiento en la misma dirección debido a que no contempla el tener que cambiar de orientación. Por otro lado, el método basado en aprendizaje por refuerzo obtiene unos resultados superiores al anterior durante esta aplicación, debido a que soluciona los problemas provocados por la falta de potencia del primero, descritos anteriormente. Con este, se puede asegurar la orientación de llegada y se puede planificar un camino que minimice la distancia recorrida en retroceso mientras se tiene en cuenta el cambio de orientación.

Es por esto que se optó por el aprendizaje por refuerzo para resolver este problema, y ha sido un éxito en cuanto a resultados, creando una apiladora autónoma capaz de moverse por una nave con 564 palés y entorno a 19600 tareas posibles que está operativa en estos momentos sin fallos reportados por parte del cliente.

Como futura línea de trabajo, se podría realizar el control óptimo del movimiento de la apiladora mediante aprendizaje por refuerzo. De este modo, cuando los objetivos están muy cercanos o la máquina está en una zona del entorno con poco espacio, la máquina podría maniobrar de forma óptima, mejorando así la capacidad de navegación de esta.

Este proyecto me ha permitido acercarme al aprendizaje por refuerzo y comprobar cómo una solución basada en él supera a otra que yo hubiera podido proponer. También me ha

permitido formar parte de un proyecto de ingeniería y aprender un poco más sobre normativas de seguridad, SLAM, bases de datos, comunicaciones TCP/IP, redes y sistemas de control de versiones, dándome una ligera idea de cómo se utilizan y cómo funcionan.

Bibliografía

Alpaydin, E. (2020). *Introduction to machine learning.* The MIT Press.

Bellman, R. (1957). *Dynamic programming.* Princeton University Press.

Cardenas, M. E., Medel, R., Castillo, J. J., Vázquez, J. C., y Casco, O. (2015). Modelos de aprendizaje supervisados: aplicaciones para la predicción de incendios forestales en la provincia de córdoba. En *XVII workshop de investigadores en ciencias de la computación.*

Hasselt, H. V. (2010). Double q-learning. En J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, y A. Culotta (Eds.), *Advances in neural information processing systems 23* (pp. 2613–2621). Curran Associates, Inc. Descargado de `http://papers.nips.cc/paper/3964-double-q-learning.pdf`

Kaelbling, L. P., Littman, M. L., y Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research 4*, 237-285.

Kober, J., y Peters, J. (2014). Reinforcement learning in robotics: A survey. En *Learning motor skills: From algorithms to robot experiments* (pp. 9–67). Cham: Springer International Publishing. Descargado de `https://doi.org/10.1007/978-3-319-03194-1_2` doi: 10.1007/978-3-319-03194-1_2

Marsland, S. (2015). *Machine learning: An algorithmic perspective* (Second ed.). CRC Press.

Martinez-Marin, T., y Duckett, T. (2005). Fast reinforcement learning for vision-guided mobile robots. En *Proceedings of the 2005 ieee international conference on robotics and automation* (p. 4170-4175).

Moreno, A., Armengol, E., Béjar, J., Belanche, L., Cortés, U., Gavaldà, R., … Sànchez, M. (1994). *Aprendizaje Automático.* Edicions UPC.

Peláez, N. (2012). *Aprendizaje no supervisado y el algoritmo wake-sleep en redes neuronales* (Tesis Doctoral no publicada). Universidad Tecnológica de la Mixteca.

Qing, G., Zheng, Z., y Yue, X. (2017). Path-planning of automated guided vehicle based on improved dijkstra algorithm. En *2017 29th chinese control and decision conference (ccdc)* (p. 7138-7143).

Safavian, S. R., y Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics*, *21*(3), 660-674.

Sutton, R. S., y Barto, A. G. (2018). *Reinforcement learning: An introduction* (Second ed.). The MIT Press.

Valeev, S., y Kondratyeva, N. (2016). Large scale system management based on markov decision process and big data concept. En *2016 ieee 10th international conference on application of information and communication technologies (aict)* (p. 1-4).

A. Salida del planificador

A.1. J/1/9_K/1/6

Acciones

A: avance

R: retroceso

'/': avance dcha

'\': avance izda

'»': retroceso dcha

'«': retroceso izda

Ruta optima (126 steps)

J/9 (O) a=\

nodo 8

C/1 (S) a=A

C/2 (S) a=A

C/3 (S) a=A

C/4 (S) a=A

C/5 (S) a=A

C/6 (S) a=A

C/7 (S) a=A

C/8 (S) a=A

C/9 (S) a=A

C/10 (S) a=A

C/11 (S) a=A

C/12 (S) a=A

C/13 (S) a=A

C/14 (S) a=A

C/15 (S) a=A

C/16 (S) a=A

C/17 (S) a=A

C/18 (S) a=\

nodo 5

L/10 (E) a=A

L/9 (E) a=A

L/8 (E) a=A

L/7 (E) a=A

L/6 (E) a=\

nodo 4

E/18 (N) a=A

E/17 (N) a=A

E/16 (N) a=A

E/15 (N) a=A

E/14 (N) a=A

E/13 (N) a=A

E/12 (N) a=A

E/11 (N) a=A

E/10 (N) a=A

E/9 (N) a=A

E/8 (N) a=A

E/7 (N) a=A

E/6 (N) a=A

E/5 (N) a=A

E/4 (N) a=/

nodo 1

K/5 (E) a=R

K/6 (E) a=A

Destino alcanzado!